Loss Monitor Blocking

Local application

Wed, Mar 28, 2007

Introduction

ECool loss monitors can be influenced by the presence of Main Injector beam during a portion of the MI cycle. Local application KHZM was designed to support tight monitoring (at 1 KHz) of the loss monitors and react promptly to inhibit the electron beam when losses exceed selected limits. But the readings of some loss monitors can be affected by MI beam, such that KHZM can cause turning off the electron beam even for losses not actually caused by the electron beam. The limits must be set unnecessarily wide to avoid trips caused by MI beam, so wide that they are not helpful for monitoring electron beam losses. This note describes a method of blocking the tight monitoring of KHZM during periods of time when MI beam is likely to cause losses that make the loss monitors look bad. The approach is to use a new LA called KHZB that watches for times to switch the limits in use by KHZM. This scheme is not ideal, since reaction to real electron beam loss that occurs when the blocking is used must await a time when the more appropriate limits are switched in. But it should be better than maintaining unhelpful wide limits all the time.

Timing example

Just to provide a sense of the timing here, consider that a time when real limits can be used, since no beam exists in the Main Injector, starts at event 0x26, which marks the end of MI flattop. At some time delay after that, we can enable real limits. After a further window of time, we can replace the real limits with wide limits, since MI beam will soon be present again. As an example, the delay after the event might be 1 second; the window of time beyond that might be 0.4 seconds. With these parameters, the tight monitoring using real limits would be enabled for 0.4 seconds out of every approximate 2 second MI cycle.

Limit sourcing

Where do the limit values come from? A console application is used to deliver the limits for all 64 channels as needed by operations according to their current operational mode. This set of limits (two 2-byte values for each channel, for a total of 256 bytes) is written to memory that is used by the 1 KHz interrupt scanning under control of KHZM. The scheme herein described will thus need access to some portion of this set of limits. (At this time, there are two sequences of channels that need this special blocking treatment, one of 11 sequential channels and another of 6. This can be handled by two instances of KHZB.)

How can KHZB know what the real limits are, as opposed to the wide limits? Assume that the wide limits are the same for all channels. It can only know the real limits by checking what they presently are as known by KHZM. And these limits can be changed by operations at any time. The concern is how to avoid overwriting new limit values with wide limits.

Check before switching

Consider an approach that monitors what the current limits are immediately before any switching is made. When KHZB starts up, let it begin by waiting for the time of (event + delay + window). This is normally the time when wide limits are to be installed. But before overwriting the KHZM limits with the wide values, first check those limits currently in use by KHZM. If they are *not* the wide limits, accept them as the real limits and then overwrite wide values. By starting up in this way, we can get a copy of the real limits for the first time.

The next stage in the logic of KHZB is to await the next occurrence of (event + delay). (Note that it must be based upon a new occurrence of the event.) At this time, check the current limits in use. If they are *not* the wide values, accept them as the real limits; otherwise, write

the previous real limits. When a time of limit switching is reached, we always check currently used limit values and accept them as the real limits if they are not wide limits.

To recognize whether the current limits in use are wide values, merely check them against the values `0x8000, 0x7FFF`. Any reading always lies within this range by definition. The check for wide values can be made on an individual basis, for each loss monitor. Also, altering the limits of one loss monitor can be done by writing a single 32-bit value that includes both the lower and upper limit values. This avoids the problem of a 1 KHz interrupt occurring when one of the two limits has been changed but not the other. (An interrupt in a 68040 cpu cannot occur in the middle of an instruction cycle.)

For some loss monitors, operations has decided to specify wide limits on purpose. The above scheme would not allow limits to be set wide permanently. As a way around this, plan to use nearly-completely wide limits in this program, such as `0x800F, 0x7FFF`. Typical loss monitor readings range from about –1 volt to a few volts positive. The not-quite-extreme low limit values should make no difference, but it will help us to recognize the special wide limits used by this program. Intentional wide limits set by operations are `0x8000, 0x7FFF`.

Access to limits

Access the limits used by `KHZM` by accessing its context clock at the appropriate offset. This is done using listype# 96. The ident format includes the Local Applications Table (`LATBL`) index used by `KHZM` and the offset into its context block. The index into `LATBL` can be found using the generic name search listype#55, in which the ident specifies a search type# and an 8-character name. For the case of looking up an entry in `LATBL`, the search type# is 1 Here, the name is '`KHZM `'. The reply gives the (local) node# and the `LATBL` index. Looking up that entry in `LATBL`, one can obtain the context memory block address at offset 8. The limits can be found at offset `0x0080` into that memory block.

Other subtleties

As both `KHZM` and `KHZB` are local applications, either has the potential of being restarted. So `KHZB` should take care to always know that it has proper access to the array of currently used limit values, which means that `KHZM` is running. It should see that `KHZM` is enabled and that its `LATBL` entry has a context block address. When `KHZM` is restarted, and its context block is initialized, its array of limit values is set to impossible (lower, upper) limit pairs of (`0x7FFF, 0x8000`). If `KHZB` notices such values, it should leave them alone, since it means that operations has not yet downloaded limit values to be used.

If another occurrence of the event is seen before (event + delay + window) has been reached, but after (event + delay) has been seen, just restart the logic awaiting (event + delay) again.

Consider this state logic:

0: await clock event, then state=1.

1: await (event+delay), then state=2 and switch to real limits.

2: await (event+delay+window), state=0 and switch to wide limits.

Start with state 0, but do not switch to real limits before we have real limits. And if we see a new event occurrence when in state 2, switch to state 1. Allow for processing both state 0 and state 1 in a single 15 Hz execution of `KHZB`, as the delay parameter may be zero. Since LAs are called at 15 Hz, the time of the limit switching may be late by as much as 66 ms. This should be kept in mind when choosing values for the delay and window parameters.

Parameter layout

Param		*Size*	*Meaning*
ENABLE	B	2	Enable Bit# for LA
LOSSMON	C	2	Loss monitor Chan#
NCHANS		2	Number of LMs
EVENT		2	Clock event#
DELAY	C	2	Delay after event Chan#
WINDOW	C	2	Window after event + delay Chan#

Analog channel parameters are used for specifying the two times, where the raw values are in milliseconds. This gives a maximum value of delay or window of about 32 seconds. The event parameter is not expected to be changed, so it is a raw parameter value.

Diagnostics

We might count the number of times we go through the motions of switching to real limits as well as the number of times we switch to wide limits. These counts should be equal.

We can keep a count of the number of channels "in play," by which is meant that number of channels actually undergoing limit value switching.

Post-implementation notes

During initialization, data requests are used to seacrch for a remote instance of KHZM in LATBL, then make a local request for the contents of that LATBL entry. In normal operation, every time it decides to switch to/from the real limit values, the Switch routine calls GetDat to update the local copy of the LATBL entry. In that way, it can ensure that the entry is still active, and it can check that it has a valid pointer to the static memory block used. That gives access to the current limit values in use by KHZM.

If a termination call is made to KHZB, then the active local data request is terminated, and the limits are switched to the real limits, before its static memory block is released.

Refinement installed Mar 27, 2007

The latest refinement was to monitor the occurrence of event 0x28 during the time that the program awaits for the time of (event + delay + window), when the tight limits are in effect. Event 0x28 occurs about 30 ms ahead of the Booster reset event that presages 8 Gev beam to be delivered to the Main Injector. Since node05E8 is running with a Micro-P Start delay of 14 ms, and Booster extraction occurs about 35 ms after reset, this means the switch to wide limits will occur about 20 ms ahead of injection into the Main Injector of 8 GeV beam extracted from the Booster.

Monitoring the switching action after this latest refinement, there are times when the tight monitoring is in effect for only one 15 Hz cycle. But at least the tight limits are used at *some* times. Before installing KHZB, extra wide limits would have to be used all the time in order to avoid the trips caused by MI beam losses nearby.